Building XML Web Services with PHP NuSOAP

Author: Ahm Asaduzzaman
Date Added: 06th Fed 2003
Type: Tutorial
Rating:

Page 1: Introduction

In this article we will first try to define Web Services, their advantages and very basic architecture. Then we will walk through two examples, showing how to create Web Services with the PHP NuSoap toolkit and how to invoke that service, which was developed in Visual Basic 6.0. In the second example we show how to use Web Services with PHP. These examples demonstrate interoperability of Web Services (platform, operating systems and language independent).

Page 2: What are Web Services?

All future software applications will be Web applications, which will run over the Internet as services for clients. Future applications will not contain large masses of compiled executable code (EXE, DLL, OCX, etc.). Applications will be broken down into a number of smaller individual services that are easier to create and easier to maintain. Individual services will be developed and maintained by smaller groups of people. It is time to wake up and get rid of our traditional skills. Find out how to create future Internet applications with the tools that are available today.

So, what are Web Services? If you ask this question to five IT gurus, you may get six different correct definitions, however, a Web service is any piece of software that makes itself available over the Internet and uses a standardized XML messaging system.

XML lies in the core of Web services, which was designed to describe data. The main functions of XML are inter-application communication, data integration and external application communication with outside partners. By standardizing on XML, different applications can more easily talk to one another, and this makes software a whole lot more interesting. If you want to know about XML, read this XML tutorial, or if you want a more intuitive feel for Web Services, try out the IBM Web Service Browser, available on the IBM Alphaworks site. The browser provides a series of Web Services demonstrations. Find more resources here.

Benefit of Web Services

Web Services let applications share data and—more powerfully—invoke capabilities from other applications without regard to how those applications were built, what operating system or platform

they run on, and what devices are used to access them. While XML Web Services remain independent of each other, they can loosely link themselves into a collaborating group that performs a particular task. Read here to learn more about benefits of different Web Services.

Evaluation of Web Services

Web Services didn't just exist suddenly. They aren't a new revolution; rather they have to be seen as an evolution based on existing Internet protocols. They are the logical next step. Since 1994, Distributed Objects have been developed by several organizations under various names. NeXT called them Portable Distributed Objects, Microsoft called them Component Object Model (COM), IBM called them System Object Model (SOM), and Apple called them OpenDoc. These companies (with the exception of Microsoft) formed the Object Management Group (OMG) and converged upon a standard called Common Object Request Broker Architecture (CORBA). The concept was to develop an architecture that lets applications plug in to an "application bus" and call a service. Based on object-oriented units, it called on specialized software components. Today, interoperable objects are being called Web Services. You can read a good introductory article here.

Web Services Protocol Stack

The Web Services protocol stack consists of three key XML-based technologies: SOAP (Simple Object Access Protocol), WSDL (Web Service Description Language), and UDDI (Universal Description, Discovery and Integration).

UDDI: This layer is responsible for centralizing services into a common registry, and providing easy publish/find functionality. Currently, service discovery is handled via the UDDI. The UDDI specification can help to solve making it possible to discover the right services from the millions currently online. Learn more about UDDI here.

WSDL: This layer is responsible for describing the public interface to a specific Web service, which is an XML-based language that describes the various functions that a Web service is capable of. Currently, service description is handled via the WSDL. You do not need to learn all the nitty-gritty details because there are tools that generate WSDL for you. You can get one here.

SOAP: This layer is responsible for encoding messages in a common XML format so that messages can be understood at either end. Currently, this includes XML-RPC and SOAP. For example, the client program bundles up two values (see Figure 1) to be added into a SOAP message, which is sent to the Web service by sending it as the body of an HTTP POST request. The server unpacks the

SOAP request that the application can understand and executes Add operation. Next, the server packages up that result of summation as response into another SOAP message, which it sends back to the client program in response to its HTTP request. The client program unpacks the SOAP message to obtain the results of the summation. So, SOAP=XML + HTTP.

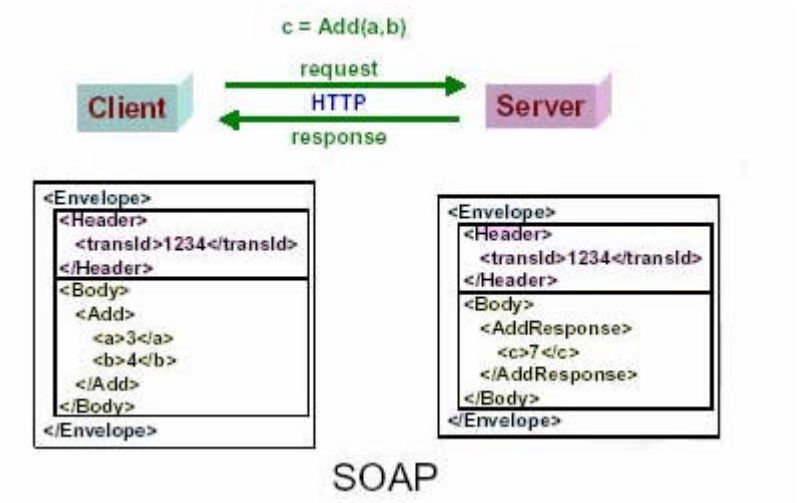

Figure 1

You can experiment with Web Services using PHP by writing only a few lines of code, without having to set up a special environment. PHP is already broadly deployed for data-centric web applications. Your PHP-driven web sites may have component that may be reused by exposing their methods using SOAP or XML RPC and this conversion process is trivial using the tools available today. For the sake of simplicity we will narrow down our discussion about SOAP. In the following example, we will use NuSOAP, which is a component-based Web Services toolkit that allows user to send and receive SOAP message over HTTP. NuSOAP is distributed by NuSphere Corporation. It is open source and licensed under the GNU LGPL.

Page 3: Walkthrough Example 1

The installation of NuSOAP is a straightforward process. Download the files and extract the file nusoap.php from the zip. For easy access, copy the classes to a location in your include path, or into the directory (say, http://yourServer.com/nusoap/) in which you will be using the classes. Let's develop a simple SOAP server with NuSOAP toolkit that will provide update information (whether a new update is released) of your software product, which is installed in the customer's computers.

Listing: cvs.php

Create a file (in this example, cvs.php) and copy it to location http://yourServer.com/nusoap/

```
<?php

require_once('nusoap.php'); // include the NuSOAP classes

$s = new soap_server; // instantiate the server object, provided by soap_server classes

$s->register('cvs'); /* to allow our function to be called remotely, we must register it with
the server object. If this is not done server will generate a fault indicating that the service
is not available if a client accesses the service. In the absence of such a registration
process, any PHP functions would be remotely available, which would present a serious security
risk. */

function cvs ($StrBuild){ /* Now define our simple function that we are exposing as service.
Notice that we first check to make sure a string (version info) was passed. If the parameter is
not a string, we use the soap_fault class to return an error to the client. */

// Optionally catch an error and return a fault

if($StrBuild == ''){

return new soap_fault('Client','','Must supply a valid build string.');

}

if ($StrBuild < "200212") { // If customer is using an older version of the product

return "A new version of this product is available now for download, please visit
http://www.wordware.com/education";

}

if ($StrBuild =="200212") {

return "Update Not Necessary. Thank you for using our product.";

}

}

$s->service($HTTP_RAW_POST_DATA); /* The final step is to pass any incoming posted data to the
SOAP server's service method. The method processes the incoming request and calls the
appropriate function (in our case cvs ()). It will then formulate the response. */
```

Now how your product will use this service, which will automatically notify your customer if a new version of your application is available. For simplicity sake we will design the following dialog with VB. You can download the project file. You will find enough commentary with code, which describes how to build SOAP envelope and parsing XML document with Microsoft XML parser.

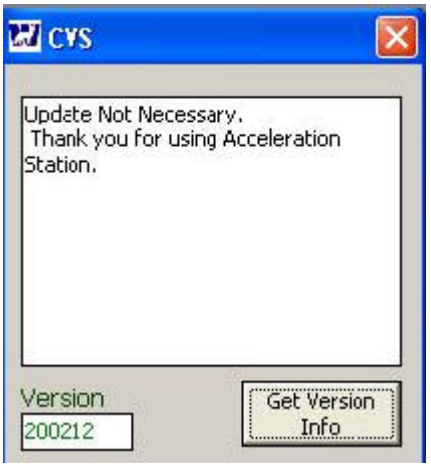

Figure 2.1

Figure 2.2

Page 4: Walkthrough Example 2

Create a Web Service that will provide a list of all ATM machines for a given zip code and develop a web site with PHP that consumes this service. Upon providing a ZIP code, user can get a list of ATM services (Figure 3).

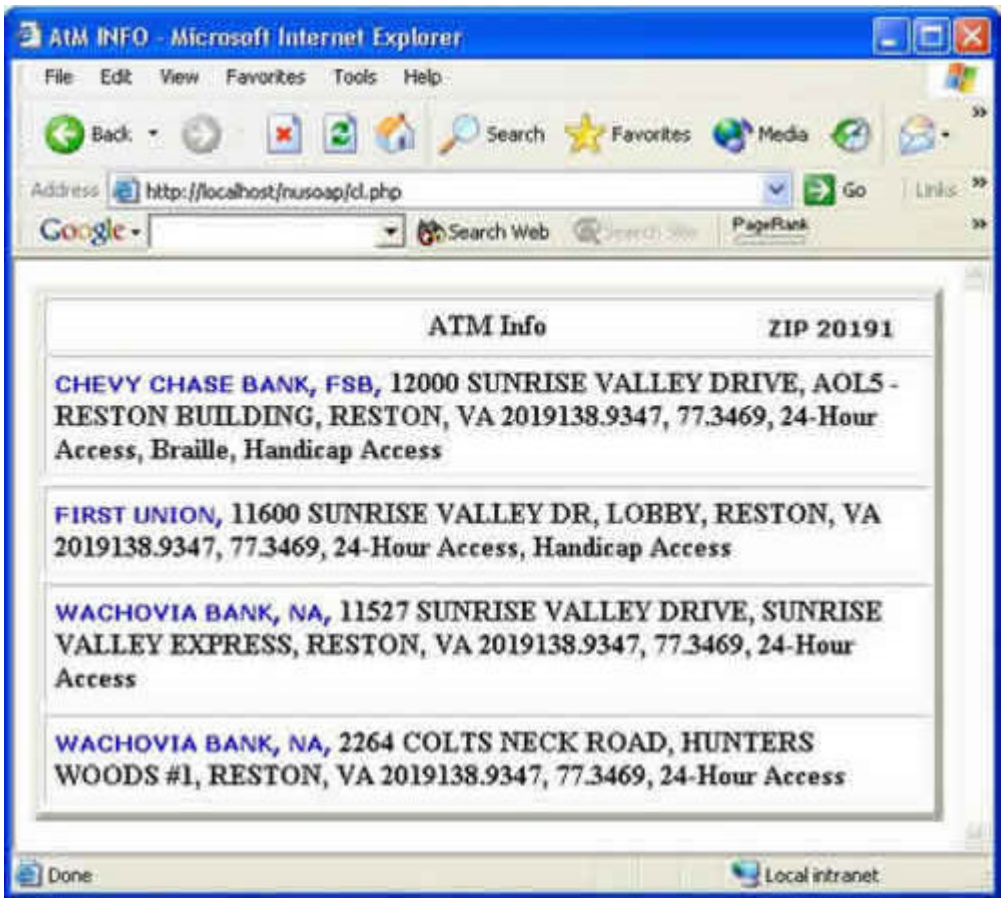

Figure 3

You can find a Web Service named GeoCash, which provide similar services. You may use this service for development purpose for 60 days with a trial key, for trial key check here. Let's walk through an example, which consumes (client.php) this service.

Step 1: Check WSDL file (if any) for this service. Prepare a request for the service with ZIP code, i.e., make a SOAP envelope (use PHP NuSOAP)

Step 2: Get the response (XML document) and parse it with XML parser PHP EXPAT.

Step 3: Display data (Figure 3)

Listing: Client.php

```
<html>
<head>
<title> AtM INFO</title>
<body>
<?
$xml_file = ‘az.xml’;
require(‘nusoap.php’);
// Prepare SOAP envelop requesting the service
$wsdlfile="http://ws2.serviceobjects.net/gc/GeoCash.asmx?WSDL";
$msg='<GetATMLocations xmlns="http://www.serviceobjects.com/"><PostalCode>20191</PostalCode>
<LicenseKey>WS2-BPN4-AFV4</LicenseKey></GetATMLocations>';
$s=new soapclient($wsdlfile,'wsdl');
// Invoking operation
$s->call('GetATMLocations',array($msg));
// Get the response
$theXML= $s->response;
// To see the response, uncomment the following lines
//echo ‘<xmp>’.$theXML.’</xmp>’;
echo "<table border=5 cellpadding=5>"; echo "<tr><th colspan=2>ATM Info</th></tr>";
// write this response to a file for future pursing
$fp = fopen ($xml_file, "w");
fwrite($fp, $theXML);
fclose($fp);

//Delete 1st 8 Lines which is not a part of XML file
$filename = $xml_file;
$fd = fopen ($filename, "r");
```

```
$query_str="HTTP/1.1 200 OK";
$m_line=7;$tmp_line=0;$line=0;
$m_filename = "result2.txt";
$m_fd = fopen($m_filename, "w");
while (!feof ($fd))
{
  $buffer = fgets($fd, 4096);
  $pos = strpos ($buffer, $query_str);

  if ($pos === false and $line==0)
  {
       fputs ($m_fd,$buffer);
  }
else
  {
     if ($tmp_line < $m_line)
     {
       $tmp_line = $tmp_line + 1;
       $line = 1;
    }
     else
     {
      $line = 0;
     }
  }
}
```

```
fclose($fd);

fclose($m_fd);

copy("result2.txt","result.txt");

$xml_file=$m_filename;



//create xml parser object

$parser = xml_parser_create();

/* Define what to do when the parser encounters a start or end tag. Note that “startElement” and “endElement” are user-defined functions, which we will look at in a minute. You can name them whatever you want, but these names are the standard convention. */

xml_set_element_handler($parser,"startElement","endElement"); xml_set_character_data_handler ($parser, "characterData");

if (!($filehandler = fopen($xml_file, "r"))) {

  die("could not open XML input");

  }



/* We start reading the contents of the file, 4K at a time, and put it in the variable “$data” until we reach the end of the file. We use xml_parse to parse each chunk as we go. */



while ($data = fread($filehandler, 4096)) {

  if (!xml_parse($parser, $data, feof($filehandler))) {

    die(sprintf("XML error:%s at line %d",

      xml_error_string(xml_get_error_code($parser)),

      xml_get_current_line_number($parser)));

  }

}



fclose($filehandler); xml_parser_free($parser);

echo "</table>";
```

```
?>

</font>

</body>

</html>




<?php



// Functions



// When the parser encounters the start element of a tag

function startElement($parser_instance, $element_name, $attrs)

{

switch($element_name) {

case "BANK" : echo "<tr><td><b><font face=verdana size=2 color=blue>";

   break;

case "ADDRESS" : echo "<b>";

   break;

case "LOCATION" : echo "<b>";

   break;

case "CITY" :   echo "<b>";

   break;

case "STATE" :   echo "<b>";

   break;

case "LATITUDE" : echo "<b>";

   break;
```

```
case "LONGITUDE" :echo "<b>";
   break;
case "NOTES" :  echo "<b>";
   break;
    }
  }
// When the parser encounters the data within the start and end tag
function characterData($parser_instance, $xml_data) {
  echo $xml_data;
  }
// When the parser encounters the end element of a tag
function endElement($parser_instance, $element_name) {
switch($element_name) {
case "BANK" :   echo ", </font>";
   break;
case "ADDRESS" : echo ", </b>";
   break;
case "LOCATION" : echo ", </b>";
   break;
case "CITY" :   echo ", </b>";
   break;
case "STATE" :   echo " </b>";
   break;
case "LATITUDE" : echo ", </b>";
   break;
case "LONGITUDE" : echo ", </b>";
   break;
```

```
case "NOTES" :   echo " </b><tr><td><b>";

   break;

  }

}

?>
```

Page 5: Conclusion

There you have it, Web Services in PHP. Now you can easily connect your applications and share data between them. Just imagine what's possible!

If you have any questions or comments about this article please post them here.
